Charger Active Defense – Proposal v2

Adam Brannon, Noah Sickels, William Lochte

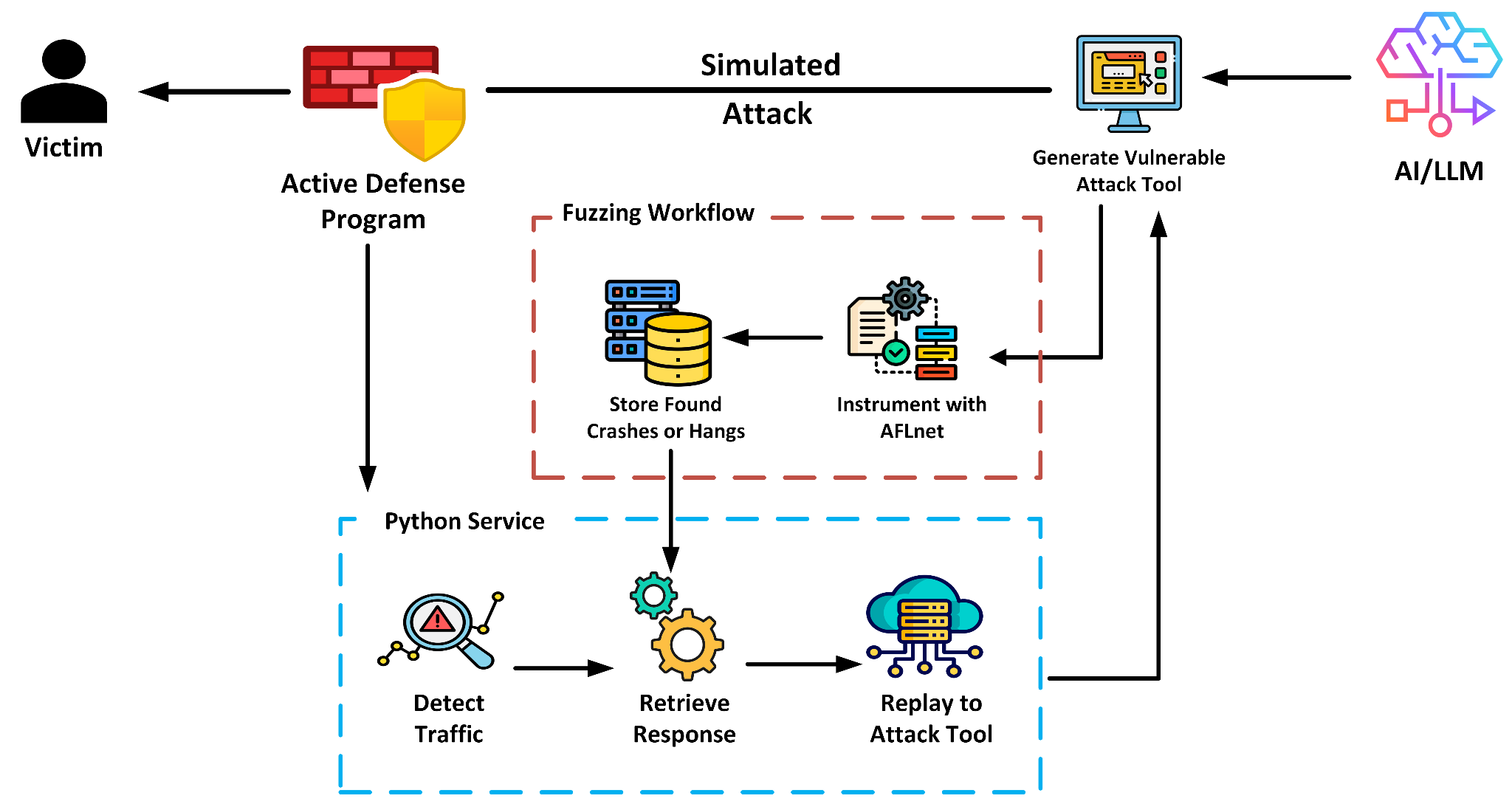
1. **Project Summary / Abstract**

The Charger Active Defense project focuses on developing a network-based fuzzing workflow to effectively and comprehensively test known and AI/LLM-generated attack tools. It aims to identify responses that may cause the attacking application to crash or hang. Responses generated could be saved and sent back to the adversary through a Python replay service during a detected attack. Due to the multi-threading nature of many attack tools like Masscan and Medusa, effectively fuzzing these tools is complicated. If a fuzzed response leads to a crash or hang, it typically occurs in a thread separate from the attacking application's instance. This crashed thread may either be reported as a false positive or not counted at all. Because of this discovery, at the customer's request, we are pivoting towards thoroughly fuzzing generated attack tools from large language models. Since we have successfully applied the fuzzing workflow to Masscan, we will also pursue the integration of ThreadSanitizer (TSan) to determine if current fuzz testing is properly crashing threads. The end goal of this workflow is to assess how we can broaden this method to apply to other attack tools, providing organizations with a strategy to defend their systems against an ever-evolving threat landscape.

1.1 The Need

Modern attack tools are highly efficient, while the cybersecurity industry struggles with active defense capabilities. Current passive defenses primarily focus on mitigating threats by disabling ports or services, which can disrupt business operations. Other methods, such as blocking source IP addresses, may prove ineffective, as adversaries can spoof IP addresses through proxies or VPNs. With the rapid advancement of artificial intelligence and prompt engineering, attackers can instantly generate various attack tools to target organizations. As AI and LLM accessibility continue to increase, developing a proactive defensive strategy has become necessary within the field to better protect our systems and limit adversaries' impact.

1.2 Charger Active Defense – Overview Diagram



2. **Survey**

2.1 Market Competition

1. Defensics, *BlackDuck (AppSec)* [1]
   1. Features: Comprehensive and flexible fuzzing tool that provides over 300 pre-built testing suites for various protocols, file formats, and interfaces. It also offers vulnerability mapping to industry standards like CWE and injection types.
   2. Price: Quote-only, est. from reference ~30 on-prem servers and licensing for $400k, ~$13k/server.
   3. Comparison:
      1. Defensics offers pre-built test suites for many different protocols. However, due to time and resource constraints, the ChAD workflow is limited to three main protocols that must be implemented by hand. Defensics does not interact with the adversarial application directly.
2. beSTORM, *BeyondSecurity* [2]
   1. Features: A black-box dynamic application security testing and fuzzing suite designed to test millions of attack combinations. Able to perform test cases without access to the source code of the targeted application. Offers 250+ pre-built modules and protocols with the ability to implement custom protocols.
   2. Price: $50,000/one-time
   3. Comparison:
      1. beSTORM provides many different protocols that are standard for fuzz tests but do not directly interact with adversarial applications. beSTORM is not open-source and requires new licensing for use with other systems.
3. Burp Suite Professional, *PortSwigger* [3]
   1. Features: A complete suite of web-based application testing tools capable of intercepting and manipulating network requests before repeating them back. Allows you to capture, filter, and query automated attack results.
   2. Price: $449/year
   3. Comparison:
      1. Burp Suite primarily focuses on fuzz testing and input validation on HTTP/HTTPS web-based traffic, but it does not support many other protocols. Burp Suite also requires a proxy such as BurpProxy or FoxyProxy to intercept incoming traffic before relaying it to the target host.

2.3 Journal Publications

A Trust Management Method Against Abnormal Behavior of Industrial Control Networks Under Active Defense Architecture [4]

* **Key Accomplishments**

1. Examined the trust management framework in industrial control networks, focusing on the security of control operations and clarifying fundamental concepts such as trust, security, and reliability in computer networks.
2. Analyzed unknown threats from Stuxnet, focusing on the operational behavior of instructions as the protected object, and extracted model trust information to identify abnormal behavior.
3. Provide a comprehensive deployment strategy for trust management, utilizing a trust server authority to facilitate network transactions.

* **Main Findings**

1. Using a trusted server authority as a mediator helps protect network hosts before malicious traffic is detected.
2. The industrial control network is vulnerable to external attacks and internal malicious behaviors, leading to abnormal actions.
3. Abnormal behavior is characterized by unauthorized access to control instructions, non-compliant operation of these instructions, and interference with normal operations.

Black-Box Fuzzing for Security in Managed Networks: An Outline [5]

* **Key Accomplishments**

1. Identify and illustrate how injection-type vulnerabilities can manifest in open service provider networks.
2. Provides a testing framework for identifying these vulnerabilities.

* **Main Findings**

1. LLDP and SNMP do not correctly sanitize incoming LLDP data despite the LLDP data standard limiting the field to alphanumeric values.
2. The fuzzer host employs a grammar-based model to generate network traffic composed of malicious LLDP frames. A probe checks whether the generated frames are processed correctly. If the LLDP service improperly handles the frames, the fuzzer stores the relevant data from the test case into a database.
3. Many black-box fuzzing tools are implemented as plug-ins.
4. The probes communicate with the fuzzer using a well-defined interface, which allows customization of the language or model used.

Rethinking Active Defense: A Comparative Analysis of Proactive Cybersecurity Policymaking [6]

* **Key Accomplishments**

1. Outline the different legal aspects of "hacking back" adversaries from a cybersecurity perspective.
2. This paper compares the efforts and outcomes of various countries regarding implementing sanctioned proactive (retaliatory) attacks on systems under threat.
3. This journal also highlights the effects of the U.S. Congress Graves bill passed in 2016.

* **Main Findings**

1. The most significant impediment to proactive defense lies in its legal and ethical nature.
2. Legal precedents on regulating defense mechanisms such as honeypots and honeynets are unclear.
3. Many possible solutions violate the United States's Computer Fraud and Abuse Act (CFAA) of 1986.

2.4 Conference Papers

Cyber Security and Defense: Proactive Defense and Deterrence [8]

* **Key Accomplishments**

1. Identifies current legal implementations for active and passive defense of networks.
2. The text examines the current landscape of "back hacking" and the counterattack strategies as part of active defense.
3. Highlights deficiencies and shortcomings in implementing proactive cyber defense measures and the lack of international cooperation.

* **Main Findings**

1. The current "back hacking" concept for proactive defense is a legal grey area.
2. Current legal measures that organizations can implement include threat monitoring and response and deception techniques using honeypot systems.

Based on Generative Adversarial Networks Seed Generation Method for Fuzzing [9]

* **Key Accomplishments**

1. Employs base64 encoding and decoding technology to expand the types of generated seeds with fuzzing tools in a flexible way.
2. Uses the RelGAN model for seed generation and subsequently modifies the generated files with a hot-spot stitching algorithm.
3. Successfully improves the performance of fuzzing on target programs with various input formats.

* **Main Findings**

1. GAN-based fuzzing of complex data with various formats improves the accuracy and quality of test input.
2. Employing a RelGAN model can allow you to generate seed files and address the imbalance between loss values and gradient updates in GAN's task of processing discrete data.
3. Combining the RelGAN model and its output with a hot-spot stitching algorithm enhances the quality of generated seeds, improving the efficiency of AFL++ in detecting crashes.

Fuzz Testing & Software Composition Analysis in Software Engineering [10]

* **Key Accomplishments**

1. Introduces Software Composition Analysis (SCA) for fuzz testing applications where the source code is available.
2. Highlights the risks associated with using open-source software for organizational use.
3. Discusses compatibility of SCA with DevOps pipelines.

* **Main Findings**

1. Open-source software results in more time and cost efficiency in application development, with higher code quality tested by a broader community.
2. Synopsys Defensics provides a testing platform for developers to discover unknown vulnerabilities proactively through mutation-based fuzz testing.

2.5 Patents

Continuous Active Defense for Digital Services [11]

* **Key Accomplishments**

1. Creating a secure client-server session method by implementing countermeasures based on predefined session-security challenge-response pairs.
2. Implement a collection of behavior patterns on the client's server to identify potential threats.
3. Implement session tracking and server response requests based on behavioral patterns and an agreed-upon client-server protocol.
4. Developed a server response system to verify the implementation of countermeasures executed by the client.

* **Main Findings**

1. This method allows for dynamic application countermeasures based on real-time behavior analysis and predefined protocols.
2. This method relies on the client device to initiate countermeasures, providing a proactive defense response.
3. The patent lists various countermeasures, including screen capture blackening, challenge responses (text completion, mouse movement, image classification), and session logout.
4. This system includes databases for storing affected sessions, countermeasures, and rules for determining the application of countermeasures based on specific triggers.

Cyber Threat Defense System and Method [12]

* **Key Accomplishments**

1. A cyber threat defense system was developed using machine learning and artificial intelligence to analyze network data and detect threats.
2. Integration of multiple machine learning modules to evaluate overall network data, identify metrics, and determine the likelihood of a breach.
3. Implemented an autonomous response module that can transmit reports and initiate mitigation actions when a breach is detected.
4. The system is designed to train its artificial intelligence classifier continually during its operational life to improve its ability to identify threats.

* **Main Findings**

1. Network data analysis can ingest and analyze network data associated with network structures and users to detect anomalies indicative of cyber threats.
2. Probability distributions for network data metrics provide a means to indicate the likelihood of an attack.
3. Machine learning models can store and generate historical network data and use it to train other models.

Fuzz Testing of Asynchronous Program Code [13]

* **Key Accomplishments**

1. Developed a fuzz testing methodology for asynchronous event processing applications.
2. Introduces the concept of setting up event processors between event sources and event sinks to allow modification of the event stream based on received test information.
3. Starts the application with asynchronous behavior, managing event sources and sinks by receiving, transforming, and supplying the modified source events to the event sink through event processors.
4. It defines numerous modules used for the methodology, including components like event abstraction, event processor, source interface, sink interface, statistical distributions, and test storage.

* **Main Findings**

1. A uniform interface connecting event processing to the sources allows abstracted event-related program code.
2. Event processors can sit between the source and sink, modifying or creating events to violate the contract and determine the application's behavior in response.
3. The statistical distribution component can store seeds used to generate particular test runs, allowing for the reproduction of identified software defects.

2.6 Other Projects

1. SnapFuzz [14]
   1. Offers a robust architecture that transforms slow asynchronous network communication into fast synchronous communications.
   2. Snapshots the target at the latest point, speeds up all file operations by redirecting them to a custom in-memory filesystem, and removes the need for many fragile modifications, such as configuring time delays or writing clean-up scripts.
   3. Integrates directly with AFLnet to improve overall performance and speed of protocol fuzzing.
2. HonggFuzz [15]
   1. Security-oriented software fuzzer that supports feedback-driven fuzzing based on code coverage for software and hardware fuzz testing.
   2. It is multi-process and multi-threaded, making it incredibly efficient. File corpus is automatically shared and improved between all fuzzed processes
   3. It provides a corpus minimization mode, enabling it to optimize input data to improve fuzzing results.
3. SGANFuzz [16]
   1. Uses Generative Adversarial Networks (GANs) to generate realistic MQTT message sequences.
   2. Supports multiple message types and parameters, allowing users to configure message sequence lengths and batch sizes.
   3. It provides visualization tools to help users analyze generated message sequences.

3. **Project Requirements**

3.1 Marketing Requirements

|  |  |
| --- | --- |
| **M1** | The ChAD fuzzing workflow must conduct network-based fuzzing to identify network responses, also known as Active Defense Responses, that can crash or hang adversarial attack tools. |
| **M2** | The existing vulnerabilities of six well-known attack tools must be documented |
| **M3** | Downselect to a single well-known attack tool for testing. |
| **M4** | Must use two different AI/LLM models to generate additional attack tools. |
| **M5** | Use each AI/LLM to generate three attack tools for fuzz testing. |
| **M6** | Software fuzzing tools capable of testing the attack tools must be identified. |
| **M7** | Demonstrate a fuzz testing workflow for Masscan and AI-generated attack tools. |
| **M8** | The ChAD program must monitor incoming network traffic to the host machine. |
| **M9** | All transaction history of incoming and outgoing response packets must be logged and recorded. |
| **M10** | The ChAD program must provide an active defense response, if found. |
| **M11** | Evaluate the effectiveness of any active defense responses found. |
| **M12** | Findings must be documented in an IEEE/ACM-style paper. |

3.2 Engineering Requirements

|  |  |
| --- | --- |
| **Marketing Requirements** | **Engineering Requirements** |
| M3 | E1: LDRA static analysis and Valgrind memory leak analysis must be used on both selected attack tools for present vulnerabilities or more favorable testing targets. |
| M6 | E2: Develop a fuzzing workflow using three fuzzing tools and approaches and rank them based on the probability of success with selected well-known attack tools. |
| M4, M5 | E3: Must use GitHub Copilot and Phind models to generate three types of attack tools each - a banner-grabber, password brute-forcer, and a simplistic multi-threaded banner-grabber. |
| M7 | E4: Must demonstrate fuzzing workflow on selected/generated attack tools |
| M1, M7, M10 | E5: The workflow must be capable of finding vulnerabilities (crashes or hangs) within attack tools, if they exist. |
| M11 | E6: We must document whether each network response crashes or hangs the attacking application and calculate and record the average hang time if it hangs the application. |
| M12 | E7: The rationale must be compiled for the selection of existing attack tools, fuzzing tools, compatibility results, and analysis for testing into an IEEE conference paper using proper formatting with font type, size, headers, and two columns per page. |
| M1, M2 | E8: Search MITRE CVE and Exploit-DB databases and compile known vulnerabilities for all six possible attack tools. |
| M8, M10 | E9: The ChAD service must send active defense responses within one minute of detection of an incoming attack. |

4. **Proposed Solution**

Our proposed solution involves utilizing two different artificial intelligence systems, Phind, and GitHub Copilot, to generate three vulnerable attack tools on which we can apply the fuzzing workflow. The three generated vulnerable attack tools will comprise a port scanning and banner-grabbing reconnaissance tool, a PostgreSQL or FTP brute-forcing tool, and a basic, multi-threaded banner-grabbing tool. This variety will allow us to test different attack methods with distinct libraries, data structures, etc.

Each program will be generated twice: once with one model and once with another. We will then compare the generated programs along with their fuzz testing results. The program most favorable in terms of compatibility and results with the fuzzing workflow will be selected as the primary fuzzing target for further testing.

If the generated tools do not produce satisfactory results, we will manually introduce vulnerabilities within the applications to assess the effectiveness of stalling attacking tools using active defense responses from fuzz testing.

The fuzzing tools AFLnet and Radamsa will remain the primary fuzz testing tools for the developed fuzzing workflow.

The current fuzz testing of Masscan with AFLnet will be paused to modify the existing workflow with the ThreadSanitizer (TSan) tool and determine whether the current workflow methodology successfully crashes the threads and not the attack tool instance itself.

We will apply the fuzzing workflow to each generated tool, ensuring it is instrumented with AFLnet and can properly run on the desired Ubuntu virtual machine against a victim virtual machine.

Once we have finalized the selection of the three attack tools, we will implement a Python service that connects to the specified IP address and port of the targeted protocol. Once it detects incoming network traffic on that port from the host, it will replay the active defense response generated from the workflow and return it to the attacking application. All transaction history of incoming and sent response packets will be logged and recorded, allowing traceability through each interaction. All research and results will be compiled into an IEEE/ACM-styled conference paper.

4.1 Current Functionality

We conducted background research by screening all six potential attack tools using LDRA Static Analysis and Valgrind to check for memory leaks. Based on these results, we selected Medusa and Masscan for further testing. We also tested the compatibility of six fuzzing tools: AFLnet, Fuzzowski, Scapy, Radamsa, Randpkt, and Peach Fuzzer. We found that all tools were compatible except for Peach Fuzzer, so we ultimately decided to focus our testing on AFLnet and Radamsa.

Additionally, we developed a workflow script that can automatically install and build our selected attack tools, fuzzing tools, and other necessary dependencies for fuzz testing. The script also includes an option to uninstall these tools. The attack tools installed by the script are Medusa and Masscan, while the selected fuzzing tools are AFLNet and Radamsa. Alternatively, we began working on a Docker implementation to provide a more streamlined solution going into the second semester.

Our rationale for attack tools, fuzzing tools, and compatibility testing results and analysis is compiled and outlined in three separate reports: G12\_fuzz\_tool\_selection\_report, G12\_attack\_tool\_selection\_report, and G12\_fuzzing\_results\_analysis.

Currently, we are running two containerized virtual machines against the Metasploitable2 VM with Masscan compiled with AFLnet.

4.1 Alternative Solutions/Tradeoffs

* Modifying Existing Attack Tools Masscan & Medusa to Remove Multi-Threading Functionality
  + This method would allow us to demonstrate the methodology of generating active defense responses in commercial or open-source applications that are more consistent with adversaries' traditional tools.
  + Masscan and Medusa have several hundred source-code files, each ranging from 30 to 400 lines long. Attempting to refactor even one application would be incredibly time-consuming and may not yield the desired results. The applications may not compile, the functionality may be broken entirely, etc. Since the principle of multi-threading is a foundation for both tools, this is not feasible given the time constraints of this semester.
* Modifying AI/LLM Generated Attack Tools to Manually Introduce Vulnerabilities
  + This method helps us more effectively test whether generating responses could cause the target application to crash or hang. By introducing manual vulnerabilities, we can examine the hypothesis that an attacking application can also become stalled similarly through fuzz testing.
  + In real-world situations, adversaries that utilize AI or large language models (LLMs) to create attack tools will not inherently introduce vulnerabilities on their own. These AI models are also less likely to adopt poor practices that lead to ineffective results. As a result, this approach could yield less accurate test cases and pretenses that this methodology works.
* Fuzzed PCAP File Generation for Fuzzing Workflow
  + This method is scriptable to generate random data packets and can be run numerous times to get different results for each pass.
  + It is easier to set up and run on any system due to minimal dependencies that are not OS/Architecturally specific.
  + Real-time protocol fuzzing is more challenging to manage but provides more accurate results about how the attack tool would react.
  + Fuzzed PCAP generation requires manually re-generating the fuzzed files each time for new test cases and does not include mutation methods to vary the tests to the same degree as live-protocol fuzzing.

5. **Testing**

5.1 Unit Tests

AI-Generated Attack Tools

* The first attack tool for port scanning instruments with AFLnet’s compiler.
* The second attack tool for banner-grabbing instruments with AFLnet’s compiler.
* The third attack tool for basic, multi-threaded port scanning instruments with AFLnet’s compiler.

Python Replay Service

* Network Handler
  + The Python service sends the active defense response to a given IP address and service port.
* Error Handling
  + If an error occurs, the Python service will log the error with a time and date.
* Retrieving Active Defense Responses
  + The Python service can successfully retrieve the required Active Defense Response from the database.
* Logging
  + The service will log all active defense responses sent to the target.
  + The history of all transactions, including time and date, will be logged.
* Resource Management
  + Once an active defense response is sent, the Python service will return to the state it was in prior to the attack.

Fuzzing Workflow

* Application to Adversarial Attack Tools Masscan and Medusa
  + The fuzzing workflow can be applied to Masscan to create a framework for fuzz testing other offensive network-based tools.
* Masscan successfully compiles with ThreadSanitizer (TSan) integration.

5.2 Integration Tests

* The fuzzing workflow can be applied to Masscan to generate potential active defense responses.
* The fuzzing workflow can be applied to the three AI/LLM-generated vulnerable applications.
* Masscan can properly instrument with AFLnet with the ThreadSanitizer module.
* The Python service can send and receive TCP and UDP network traffic to and from the attacking virtual machine.
* Once traffic is detected, the Python service can send the Active Defense Response to the specified attacking machine’s IP address and port.

5.3 Acceptance Tests

Python Replay Service

* The Python service detects incoming network traffic from the specified port and protocol that is being attacked by the AI-generated attack tools.
* The Active Defense Response is sent to the attacking machine’s service.
* If a vulnerability is discovered, the attack tool crashes or hangs when supplied with an active defense response.

Fuzzing Workflow

* A workflow framework is provided through the IEEE conference paper and detailed documentation.
* Workflow can be applied to numerous offensive security tools - known or AI-generated.
* Workflow can be applied using various fuzzing tools.

6.0 **Project Timeline & Milestones**

6.1 Milestones

|  |  |  |  |
| --- | --- | --- | --- |
| **Milestones** | **Primary** | **Secondary** | **Dependencies** |
| 1.0 Gen-AI Banner-Grabbing Attack Tool | Noah | Adam | - |
| 2.0 Compare Gen-AI Banner-Grabbing Tool Performance Between Models | Adam | Will | 1.0 |
| 3.0 LDRA Static Analysis on Gen-AI Banner-Grabber Tools | Will | Adam | 1.0 |
| 4.0 Masscan ThreadSanitizer Integration for AFLnet Fuzzing | Noah | Will | - |
| 5.0 Gen-AI Brute-Force Attack Tool | Will | Noah | 1.0, 2.0, 3.0 |
| 6.0 Compare Gen-AI Brute-Force Attack Tool Performance Between Models | Noah | Adam | 5.0 |
| 7.0 LDRA Static Analysis on Gen-AI Brute-Force Tools | Adam | Will | 5.0 |
| 8.0 Python Relay Network Handling Module | Noah | Will | - |
| 9.0 Python Relay Error and Traffic Logging Module | Will | Noah | 8.0 |
| 10.0 Gen-AI Multi-Threaded Banner-Grabbing Attack Tool | Noah | Adam | 5.0, 6.0, 7.0 |
| 11.0 Compare Gen-AI Multi-Threaded Banner-Grabbing Attack Tool Performance Between Models | Will | Noah | 10.0 |
| 12.0 LDRA Static Analysis on Gen-AI Multi-Threaded Banner-Grabber Tools | Adam | Will | 10.0 |
| 13.0 Review of Masscan ThreadSanitizer Results | Noah | Adam | 4.0 |
| 14.0 Unit Tests on AI-Generated Attack Tools | Noah | Will | 1.0, 5.0, 10.0 |
| 15.0 Unit Tests of Python Replay Service | Will | Adam | 8.0, 9.0 |
| 16.0 Integration Tests | Adam | Noah | 14.0, 15.0 |
| 17.0 Acceptance Tests for Python Replay Service | Noah | Will | 14.0, 15.0, 16.0 |
| 18.0 Acceptance Tests for Fuzzing Workflow | Will | Adam | 4.0, 13.0, 17.0 |
| 19.0 IEEE Conference Paper | - | - | 2.0, 3.0, 6.0, 7.0, 11.0, 12.0 |

6.2 Activities

|  |  |
| --- | --- |
| **Related Milestone** | **Activity** |
| 1.0 Gen-AI Banner-Grabbing Attack Tool | 1.1 Define requirements for the banner-grabbing tool. |
|  | 1.2 Generate banner-grabber from the GitHub Copilot model. |
|  | 1.3 Generate banner-grabber from the Phind model. |
|  | 1.4 Compile and instrument with AFLnet. |
|  | 1.5 Test generated tool on Metasploitable2 VM. |
| 2.0 Compare Gen-AI Banner-Grabbing Tool Performance Between Models | 2.1 Define metrics for comparison (speed, accuracy, libraries and data structures used, etc.) |
|  | 2.2 Collect fuzz testing and compatibility results with existing workflow. |
|  | 2.3 Compare the code structure and efficiency between the two different models. |
|  | 2.4 Document differences, accuracy, efficiency, and results between models. |
| 3.0 LDRA Static Analysis on Gen-AI Banner-Grabber Tools | 3.1 Compile source code on the Blackhawk server. |
|  | 3.2 Run TestBed Quality Review tests on the GitHub Copilot model code. |
|  | 3.3 Run TestBed Quality Review tests on the Phind Model code. |
|  | 3.4 Document and save generated reports for IEEE conference paper. |
| 4.0 Masscan ThreadSanitizer Integration for AFLnet Fuzzing | 4.1 Add ThreadSanitizer hook module to current existing workflow |
|  | 4.2 Re-compile and instrument Masscan with added ThreadSanitizer module |
|  | 4.3 Start fuzz testing with Tsan integration with AFLnet |
| 5.0 Gen-AI Brute-Force AttackTool | 5.1 Define requirements for brute-force tool. |
|  | 5.2 Generate brute-force tool from the GitHub Copilot model. |
|  | 5.3 Generate brute-force tool from the Phind model. |
|  | 5.4 Compile and instrument with AFLnet. |
|  | 5.5 Test generated tool on Metasploitable2 VM. |
| 6.0 Compare Gen-AI Brute-Force Attack Tool Performance | 6.1 Define metrics for comparison (speed, accuracy, libraries and data structures used, etc.) |
|  | 6.2 Collect fuzz testing and compatibility results with existing workflow. |
|  | 6.3 Compare the code structure and efficiency between the two different models. |
|  | 6.4 Document differences in accuracy, efficiency, and results between models. |
| 7.0 LDRA Static Analysis on Gen-AI Brute-Force Tools | 7.1 Compile source code on the Blackhawk server. |
|  | 7.2 Run TestBed Quality Review tests on GitHub Copilot model code. |
|  | 7.3 Run TestBed Quality Review tests on Phind Model code. |
|  | 7.4 Document and save generated reports for IEEE conference paper. |
| 8.0 Python Relay Network Handling Module | 8.1 Implement network traffic receiving capability by connecting to the network interface. |
|  | 8.2 Configure database hook for retrieving active defense responses from fuzzing workflow. |
|  | 8.3 Implement network traffic sending capability to given host address and port. |
| 9.0 Python Relay Error and Traffic Logging Module | 9.1 Create logger schema for formatting |
|  | 9.2 Implement network traffic logging from the network handler module |
|  | 9.3 Implement error logging on exception |
|  | 9.4 Add sent active defense response logging |
| 10.0 Gen-AI Multi-Threaded Banner-Grabbing Attack Tool | 10.1 Define requirements for the multi-threaded banner-grabbing tool. |
|  | 10.2 Generate multi-threaded banner-grabber from the GitHub Copilot model. |
|  | 10.3 Generate multi-threaded Banner-Grabber from the Phind model. |
|  | 10.4 Write ThreadSanitizer hook. |
|  | 10.5 Compile and instrument with AFLnet. |
|  | 10.6 Test generated tool on Metasploitable2 VM. |
| 11.0 Compare Gen-AI Multi-Threaded Banner-Grabbing Attack Tool Performance | 11.1 Define metrics for comparison (speed, accuracy, libraries and data structures used, etc.) |
|  | 11.2 Collect fuzz testing and compatibility results with existing workflow |
|  | 11.3 Compare the code structure and efficiency between the two different models |
|  | 11.4 Document differences, accuracy, efficiency, and results between models. |
| 12.0 LDRA Static Analysis on Gen-AI Multi-Threaded Banner-Grabber Tools | 12.1 Compile source code on the Blackhawk server. |
|  | 12.2 Run TestBed Quality Review tests on the GitHub Copilot model code. |
|  | 12.3 Run TestBed Quality Review tests on the Phind Model code. |
|  | 12.4 Document and save generated reports for IEEE conference paper. |
| 13.0 Review of Masscan ThreadSanitizer Results | 13.1 Collect fuzz testing results from Masscan with ThreadSanitizer. |
|  | 13.2 Analyze whether crashed threads occurred. |
|  | 13.4 Add ThreadSanitizer to existing fuzzing workflow. |
| 14.0 Unit Tests of AI-Generated Attack Tools | 14.1 GitHub Copilot’s banner-grabbing tool compiles with afl-gcc. |
|  | 14.2 Phind’s banner-grabbing tool compiles with afl-gcc. |
|  | 14.3 GitHub Copilot’s brute-force tool compiles with afl-gcc. |
|  | 14.4 Phind’s brute-force tool compiles with afl-gcc. |
|  | 14.5 Copilot’s multi-threaded banner-grabbing tool compiles with afl-gcc. |
|  | 14.6 Phind’s multi-threaded banner-grabbing tool compiles with afl-gcc. |
| 15.0 Unit Tests of Python Replay Service | 15.1 UDP traffic can be sent to the attacking virtual machine. |
|  | 15.2 TCP traffic can be sent to the attacking machine |
|  | 15.3 Network traffic history (packets sent, received, etc.) logged properly. |
|  | 15.4 Error and exceptions are properly logged. |
| 16.0 Integration Tests | 16.1 Fuzzing workflow applies to Masscan to generate active defense responses. |
|  | 16.2 Fuzzing workflow applies to all three AI-generated attack tools. |
|  | 16.3 Masscan can properly instrument with AFLnet with the ThreadSanitizer hook. |
|  | 16.4 The Python service can send and receive TCP and UDP network traffic to and from the attacking virtual machine. |
|  | 16.5 The Python service can send the active defense response to the specified attacking machine’s IP address and port. |
| 17.0 Acceptance Tests for Python Replay Service | 17.1 The Python service detects incoming traffic from the specified port and protocol under attack from the AI-generated attack tools. |
|  | 17.2 The active defense response is sent to the attacking machine’s service. |
|  | 17.3 If a vulnerability is found, the attack tool crashes or hangs when supplied with an active defense response. |
| 18.0 Acceptance Tests for Fuzzing Workflow | 18.1 The fuzzing workflow framework is provided through the IEEE conference paper and detailed documentation. |
|  | 18.2 The workflow can be applied to numerous offensive security tools - known or AI-generated. |
|  | 18.3 The workflow can be applied using various fuzzing tools. |
| 19.0 IEEE Conference Paper | 19.1 Compile all reports for attack tool selection, fuzz tool selection, and compatibility testing from 1st semester. |
|  | 19.2 Compile all reports to compare and analyze each model's three generated attack tools. |
|  | 19.3 Compile all LDRA and Valgrind reports from each attack tool. |
|  | 19.4 Compile all results into final IEEE conference paper format. |

7. **Relevant Environmental/Societal Impact of Project**

7.1 Environmental Impact

* Energy Consumption
  + The fuzzing workflow demands substantial computational resources to operate the fuzzing tools and analyze network responses. The process involves several virtual machines running concurrently for several months at a time. This higher energy consumption contributes to greenhouse gas emissions, as much of our energy production relies on fossil fuels. To address this issue, we can investigate using energy-efficient hardware and optimize software to minimize computational overhead.
* Electronic Waste (E-Waste)
  + Frequent upgrades and hardware replacements required to maintain security updates can increase electronic waste. Proper disposal methods, including recycling programs and e-waste management, should be implemented to minimize environmental impact. Additionally, extending the lifespan of hardware through regular maintenance and updates can help reduce the need for replacements.
* Raw Material Procurement
  + The raw materials for the hardware used in the project are typically procured through mining and manufacturing processes that can adversely affect wildlife, forests, and water quality. Sustainable sourcing practices and using recycled materials can help mitigate these impacts.

7.2 Security Issues

* Vulnerabilities
  + If not properly secured, the fuzzing workflow itself could introduce vulnerabilities. Ensuring that the fuzzing tools and the active defense service application are regularly updated and patched is crucial. Implementing secure coding practices and regular security audits can help identify and address potential vulnerabilities.
  + The Metasploitable2 virtual machine is designed to be an insecure target for fuzz testing. It features multiple vulnerabilities due to outdated protocol versions and insecure configurations. As such, it will not be connected to the Internet and will be strictly limited to the internal network between the two virtual machines.
* Other Associated Risks
  + The attacking virtual machine for this project is Kali Linux 2024.1. This OS is pre-built with hundreds of attack tools designed for penetration testing or adversarial emulation. If used maliciously, it presents an immediate threat to any corporate or institutional network. This virtual machine's interaction and traffic will be carefully monitored and properly segmented from critical network traffic to ensure the safety of the network in use.

7.3 Inter-operability Issues

* Legacy Systems
  + The fuzzing workflow may need to interact with legacy systems that use outdated protocols and insecure security measures, such as outdated PostgreSQL or SSH versions. One of the attacking virtual machines is an Ubuntu 18.04 OS, which no longer receives long-term support (LTS) updates from Canonical. This system will be segmented on the network to avoid potential issues on the network.

7.4 Personal Privacy

* Data Handling
  + The project involves analyzing network traffic that may contain sensitive information. Fuzz testing is conducted on a private, configured network equipped with a firewall specifically set up for this purpose. It is crucial to ensure that personal data is anonymized and handled securely to safeguard privacy. Implementing strict access controls and encryption can help protect personal data without compromising services.
  + This project's proactive defense aspect (e.g., "back hacking") involves targeting another system from the perspective of that attacking system being foreign to the defenders. Both machines are owned and operated by us or a trusted institution in our use case. In a real-time situation where we are actively defending our system against a foreign adversary, trying to hang or crash the attacker's application would directly violate the integrity of the attacker's personal computer and the data it holds.

7.5 Health & Safety

* Developmental Safety
  + The fuzzing workflow development involves dealing with potentially harmful network traffic. It is essential to ensure that the development environment is secure and that developers are trained in safe handling practices to prevent accidental exposure to malicious data.
* Product Safety
  + The active defense fuzzing workflow must be configured and designed to avoid unintended disruptions to legitimate network traffic, which could impact the availability of critical services. Thorough testing and validation are necessary to ensure the product's safety and reliability.

7.6 Regulatory & Legal Issues

* Compliance
  + The project must comply with various regulations and standards related to cybersecurity, data protection, and electronic waste management. In the United States, this includes regulations such as the General Data Protection Regulation (GDPR) for data privacy and the Resource Conservation and Recovery Act (RCRA) for e-waste.
  + This project's proactive defense aspect (e.g., "back hacking") involves targeting another system from the perspective of that attacking system being foreign to the defenders. Both machines are owned and operated by us or a trusted institution in our use case. In a real-time situation where we are actively defending our system against a foreign adversary trying to hang or crash the attacker's application, this would directly violate the Computer Fraud and Abuse Act (CFAA) of 1986.

7.7 Trade-offs

* Environmental vs. Economic
  + Balancing environmental impact with economic considerations presents a significant trade-off. While investing in energy-efficient hardware and sustainable materials may increase initial costs, the long-term benefits, such as reduced environmental harm and compliance with regulations, can outweigh the initial cost. However, sourcing environmentally friendly hardware that meets the performance demands of virtual machines and fuzz testing applications could be problematic. Additionally, increased regulatory oversight may lead to even higher costs for that hardware.
* Health & Safety vs. Economic
  + Ensuring the health and safety of the group members may require more time and resources, as ensuring the configuration of virtual machines and applications does not compromise the integrity of the network or the machines used for testing.

7.8 Security & Risk Analysis of Components & Systems

* The fuzzing workflow and necessary software (fuzz testing tools, e.g., AFLnet, Fuzzowski, Radamsa, etc.) on the virtual machines (Kali Linux 2024.1, Ubuntu 18.04, Metasploitable2) must be configured not to disrupt regular network traffic while in use to maintain the operations of adjacent systems. Implementing robust security measures (two-factor authentication, complex password policy, etc.), regular updates on relevant hardware, and continuous monitoring to detect and respond to potential attacks is paramount. Regular backups or snapshots of the virtual machines will be taken to preserve their integrity.

8. **Conclusion**

Modern attack tools are increasingly efficient, posing challenges for the cybersecurity industry. They mainly rely on passive defense strategies that can disrupt operations. Current methods, like blocking source IP addresses, often prove ineffective. The rapid advancement of AI enables attackers to generate diverse attack tools quickly.

The proposed solution is a network-based fuzzing workflow designed to test known and AI-generated attack tools to identify responses that cause crashes or hangs in attacking applications. This process is complicated by the multi-threaded nature of many attack tools, which can lead to false positives. The focus is now on thoroughly fuzzing these generated tools and integrating ThreadSanitizer to ensure accurate crash detection. The ultimate goal is to expand this method to other attack tools, creating a proactive defense strategy as AI accessibility grows.

Appendix

[1] BlackDuck, AppSec. “Defensics Fuzz Testing Tools & Services | Black Duck.” *Blackduck.com*, 2023, www.blackduck.com/fuzz-testing.html. Accessed 10 Jan. 2025.

[2] Security, Beyond. “Dynamic Application Security Testing Tool (DAST) | BeSTORM.” *Vulnerability Security Testing & DAST | beyond Security*, 2019, www.beyondsecurity.com/products/bestorm. Accessed 10 Jan. 2025.

[3] PortSwigger. “Burp Suite Professional.” *Portswigger.net*, 2020, portswigger.net/burp/pro. Accessed 10 Jan. 2025.

[4] J. Wang, Z. Zhang and M. Wang, "A Trust Management Method Against Abnormal Behavior of Industrial Control Networks Under Active Defense Architecture," in IEEE Transactions on Network and Service Management, vol. 19, no. 3, pp. 2549-2572, Sept. 2022, doi: 10.1109/TNSM.2022.3173398.

[5] L. Fernandez and G. Karlsson, "Black-Box Fuzzing for Security in Managed Networks: An Outline," in IEEE Networking Letters, vol. 5, no. 4, pp. 241-244, Dec. 2023, doi: 10.1109/LNET.2023.3286443.

[6] Shackelford, Scott. “Rethinking Active Defense: A Comparative Analysis of Proactive Cybersecurity Policymaking.” Penn Carey Law: Legal Scholarship Repository, 2019, scholarship.law.upenn.edu/jil/vol41/iss2/3/. Accessed 10 Jan. 2025.

[7] Hilario, E., Azam, S., Sundaram, J. et al. Generative AI for Pentesting: the good, the bad, the ugly. *Int. J. Inf. Secure*. 23, 2075–2097 (2024).<https://doi.org/10.1007/s10207-024-00835-x>

[8] M. Şenol, "Cyber Security and Defense: Proactive Defense and Deterrence," 2022 3rd International Informatics and Software Engineering Conference (IISEC), Ankara, Turkey, 2022, pp. 1-6, doi: 10.1109/IISEC56263.2022.9998314.

[9] Z. Cheng, Y. Guo, X. Li and J. Hu, "Based on Generative Adversarial Networks Seed Generation Method for Fuzzing," 2024 3rd International Conference on Big Data, Information and Computer Network (BDICN), Sanya, China, 2024, pp. 97-101, doi: 10.1109/BDICN62775.2024.00025.

[10] E. Yang, "Fuzz testing & software composition analysis in software engineering," 2018 International Symposium on VLSI Design, Automation and Test (VLSI-DAT), Hsinchu, Taiwan, 2018, pp. 1-3, doi: 10.1109/VLSI-DAT.2018.8373240.

[11] Casas, Jose Carlos. Continuous Active Defense for Digital Services. 18 Jan. 2024, patents.google.com/patent/US20240022581A1/en?oq=20240022581. Accessed 10 Jan. 2025.

[12] Humphrey, Dickon Murray. *Cyber Threat Defense System and Method*. 20 Aug. 2024, patents.google.com/patent/US20210273960A1/en. Accessed 10 Jan. 2025.

[13] Meijer, Erik. Fuzz Testing of Asynchronous Program Code. 21 Apr. 2015, patents.google.com/patent/US20120089868A1/en?oq=20120089868. Accessed 10 Jan. 2025.

[14] srg-imperial. “GitHub - Srg-Imperial/SnapFuzz.” *GitHub*, 2022, github.com/srg-imperial/SnapFuzz. Accessed 10 Jan. 2025.

[15] honggfuzz, Google. “Honggfuzz.” *GitHub*, 25 Jan. 2022, github.com/google/honggfuzz. Accessed 10 Jan. 2025.

[16] PeterWeiJust. “GitHub - PeterWeiJust/SGANFuzz.” *GitHub*, 2023, github.com/PeterWeiJust/SGANFuzz. Accessed 10 Jan. 2025.